**Факултет техничких наука у Чачку**

**Универзитета у Крагујевцу**

**Иван Митровић**

**Spring Boot framework и веб сокети на примеру чет апликације**

**дипломски рад**

**Чачак, октобар 2022.**

**Факултет техничких наука у Чачку**

**Универзитета у Крагујевцу**



**Spring Boot framework и веб сокети на примеру чет апликације**

**Дипломски рад**

**Врста студија: Основне академске студије**

**Назив студијског програма: Информационе технологије**

**Предмет: Напредно објектно-оријентисано програмирање**

**Студент: Руководилац рада:**

**Никола Радовановић 67-2018 Др Олга Ристић, ванр. проф.**

**Чачак, септембар 2022.**

**Резиме**

Кључне речи:

* Java
* Spring
* Spring Boot
* Фрејмворк
* Веб
* Сокет

**Summary**

Keywords:

* Java
* Sprin
* Spring Boot
* Framework
* Web
* Socket

# Садржај

[1. Увод 5](#_Toc112165172)

[2. Spring Framework 7](#_Toc112165173)

[2.1 Креирање Spring boot пројекта 7](#_Toc112165174)

[3. План развоја апликације 9](#_Toc112165175)

[3.1 Коришћени алати и програмска подршка 9](#_Toc112165176)

[3.1.1 Spring Boot 9](#_Toc112165177)

[3.1.2 Maven 10](#_Toc112165178)

[3.1.3 XAMPP 11](#_Toc112165179)

[3.1.4 Apache Tomcat 11](#_Toc112165180)

[3.1.5 Eclipse 11](#_Toc112165181)

[3.1.6 Jasper Reports 12](#_Toc112165182)

[3.1.7 Thymeleaf 12](#_Toc112165183)

[3.2 Коришћене зависности 12](#_Toc112165184)

[3.3 Глобални опис 13](#_Toc112165185)

[3.3.1 Перспектива система 13](#_Toc112165186)

[3.3.2 Карактеристике система 13](#_Toc112165187)

[3.3.3 Класе корисника и карактеристике 13](#_Toc112165188)

[3.4 Дефинисање захтева апликације 14](#_Toc112165189)

[3.4.1 Функционални захтеви 14](#_Toc112165190)

[3.4.2 Нефункционални захтеви 15](#_Toc112165191)

[3.4.3 Процесни захтеви 16](#_Toc112165192)

[4. Архитектура апликације 18](#_Toc112165193)

[4.1 Spring Boot архитектура 18](#_Toc112165194)

[4.2 Spring Boot архитектура радног тока 20](#_Toc112165195)

[4.3 База података 25](#_Toc112165196)

[5. Изглед апликације 27](#_Toc112165197)

[6. Закључак 35](#_Toc112165198)

[7. Литература 36](#_Toc112165199)

[8. Списак слика 37](#_Toc112165200)

# Увод

Развој софтвера је једна од области која не престаје да напредује и програмери константно покушавају да пронађу нова решења која цео процес чине лакшим са једне стране, а са друге стране произведене апликације моћнијим, поузданијим и једноставнијим за коришћење. Један од новитета који је успео да задовољи обе стране јесте Spring Boot фрејмворк.

Spring Boot својим корисницима омогућава лако креирање апликација које се без проблема покрећу на серверима. Овај фрејмворк је изграђен на већ познатој Spring платформи и омогућава доста лакшу иницијализацију пројекта. Такође подржава и коришћење спољашњих библиотека. Главни циљ овог фрејмворка је да поједностави процес развоја веб апликација и знатно смањи дужину кода што је по мом мишљењу успео да испуни а и на основу његове популарности међу осталим Java фрејмворцима и остали би се сложили. У наставку приказана је листа најпопуларнијих Јава фрејмворка (<https://hackr.io/blog/java-frameworks>).

1. Spring
2. Struts
3. Hibernate
4. Apache Wicket
5. JSF (Java Server Faces)
6. Dropwizard
7. Grails
8. ATG
9. Play
10. Apache Hadoop

Као што можемо видети Spring је на првом месту и његова популарност наставља да расте што показује и чињеница да је његово коришћење у 2021. години порасло за 4% у односу на претходне године. Постоји доста разлога који су га учинили овако популарним, а неки од њих су:

* Смањује време потребно за развој и повећава целукупну ефикасност развојног тима
* Самостално конфигурише све компоненте потребне за израду апликације
* Подршка корисницима, омогућава пружање помоћи на даљину
* Интеграција са Spring екосистемом што омогућава коришћење Spring Data, Security, ORM и JDBC
* Има уграђене HTTP сервере за тестирање веб апликација као што су Tomcat и Jetty
* Лако повезивање са базама података
* Уштеда времена при писању кода.

Циљ дипломског рада је реализација веб апликације која ће кориситити Spring Boot фрејмворк и омогућити приказ његове архитектуре. Ова веб апликација функционисаће као POS (Point of Sale) систем који ће омогућити корисницима основне функције за ову врсту система (продаја, управљање подацима, генерисање фактура...) и унапредити претходне итерације сличних система које су користиле старије архитектуре и технологије. Сама апликација, њен начин функционисања и упутство за коришћење биће описане у наставку рада, такође у раду ће бити описана и Spring Boot архитектура где ће сви њени аспекти бити детаљно објашњени.

# Spring Framework

Прва верзија Spring фрејмворка објављена је од стране Рода Џонсона у публикацији своје књиге ,,*Expert One-on-One J2EE Design and Development*’’ из октобра 2002 године. Створио је Spring како би олакшао развијање Java програма са серверске стране. У употреби се по први пут нашао у јуну 2003. године под Apache 2.0 лиценцом. У периоду од 2003. године до данас објављено је више различитих верзија од којих је последња стабилна верзија 5.3.

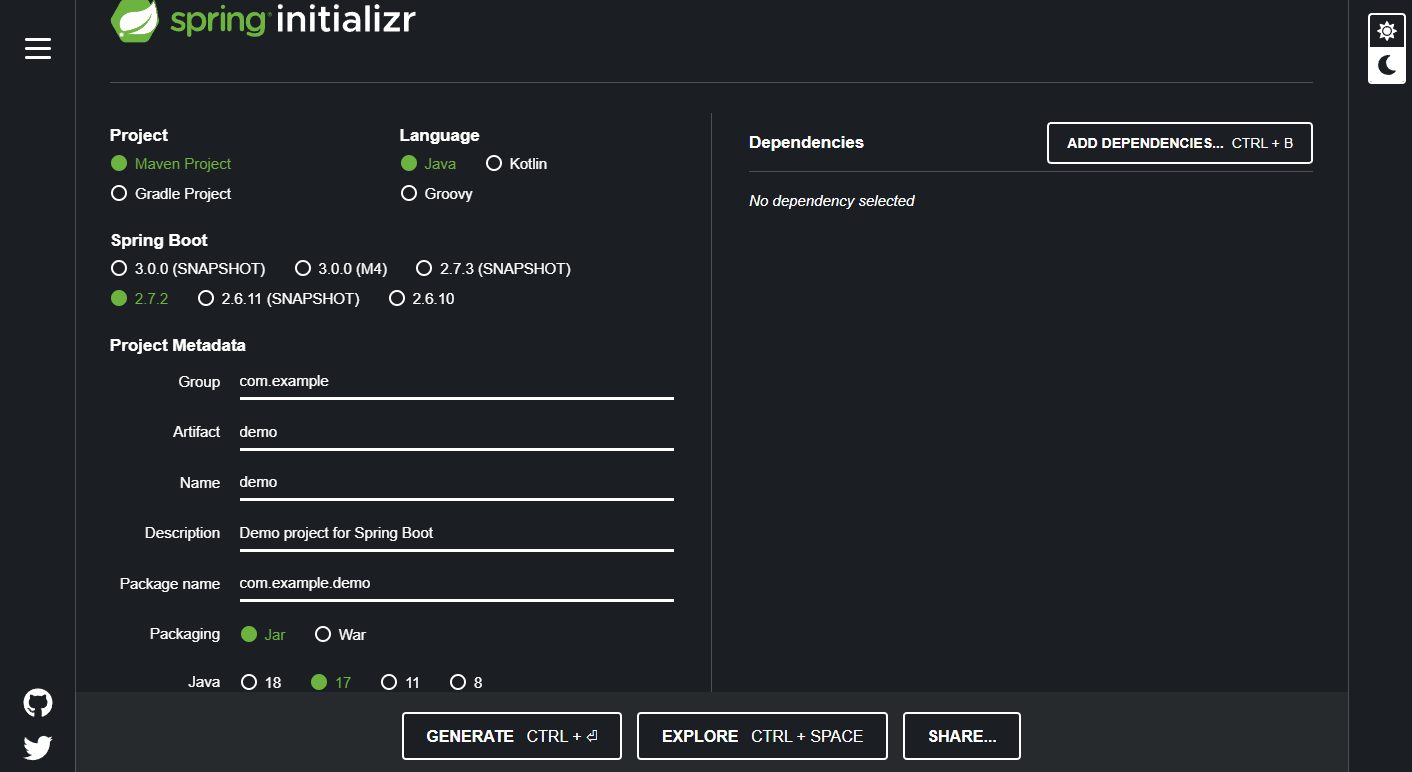
Spring Boot је настао нешто касније када је програмер по имену Мајк Јангстром оставио коментар на једном интернет форуму и предложио начин како би се постојећи фрејмворк могао побољшати и учинити једноставнијим за коришћење тако што би сви сервиси били повезани и обједињени унутар једног Spring контејнера који би био покренут из главне (main() ) методе. Тај коментар је видео Фил Веб један од чланова Spring развојног тима и недуго затим кренуо је са развијањем Spring Boot-а. Прва радна верзија објављена је у августу 2014. године и убрзо је стекла доста популарности због своје иновативности. Тренутна актуелна верзија је 2.7.0 која је коришћена за израду апликације, и објављена у мају 2022. године.

## Креирање Spring boot пројекта

За креирање Spring boot пројекта потребно је да прво додамо све потребне зависности у наш пројекат. Уколико наш програм користи неку класу кажемо да наш програм има зависност према тој класи и ту зависност потребно је дефинисати при иницијализацији. Један од најпознатијих и најједноставнијих алата за додавање зависности је Spring Initializr (<https://start.spring.io/>). У суштини Spring Initializr је веб апликација која аутоматски генерише Spring Boot структуру пројекта за нас. Не генерише код, али доста олакшава сам процес креирања пројекта. На слици 1. приказан је изглед ове веб апликације.

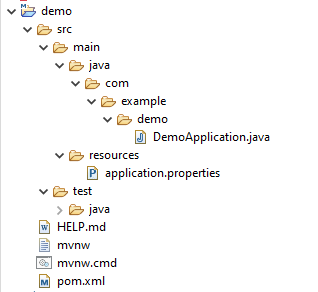
Пре него што генеришемо пројекат потребно је прво да дефинишемо неколико параметара.

У Project делу дефинишемо који тип пројекта желимо да користимо, можемо да бирамо између Maven и Gradle пројекта. Language поље нуди три опције и односи се на програмски језик у ком ће пројекат бити написан. Можемо да бирамо између Jave, Kotlin-a и Groovy. Испод тога бирамо верзију Spring Boot-a, актуелна верзија је 2.7.2. Након тога потребно је да попунимо неколико поља која се односе на именовање пројекта и различитих компоненти унутар пројекта. Initializr такође нуди и две опције за паковање пројекта, Jar или War. Затим, потребно је да изаберемо Java верзију, и на крају дефинишемо све потребне зависности које ће користити наш пројекат.



*Слика 1. Spring Initializr*

Листа зависности које можемо додати у наш пројекат је велика, а неке од битнијих су: Spring Web, Spring Security, Spring JPA… Када завршимо са навођењем потребних зависности, кликом на Generate дугме генерише се zip фајл који се аутоматски преузима и касније га је потребно увести у развојно окружење које желимо да користимо. Након импортовања, структура пројекта требало би да буде као што је приказано на слици 2.



*Слика 2. Изглед пројекта након увоза*

# План развоја апликације

Планирање је јако битна ставка у циклусу развоја сваког пројекта и програмерима омогућава ослонац када се пронађу у непознатој ситуацији где нису сигурни који је следећи корак. Зато је потребно тачно дефинисати захтеве пројекта, описати коришћен алат и друге неопходне функционалности.

## Коришћени алати и програмска подршка

Пре него што кренемо са израдом апликације потребно је да одлучимо које ћемо алате користити, проверимо њихову доступност и евентуално упознамо се са њиховим дизајном. Одабран алат и програми треба да буду одржавани и редовно ажурирани како би се избегле нежељене грешке у току израде пројекта.

### Spring Boot

Spring Framework је развојно окружење отвореног кода које се користи за развој апликација које могу самостално функционисати. Spring Boot користи Spring Framework и додатно олакшава креирање веб апликација и микросервиса кроз три кључне специфичности: аутоконфигурацију, зависни приступ и самосталност апликација.

Аутоконфигурација значи да се апликације иницијализују са већ дефинисаним зависностима које није потребно ручно наводити. Како Spring Boot већ поседује уграђене могућности за аутоконфигурацију он аутоматски конфигурише и Spring Framework, а и друге пакете који се користе унутар пројекта, а наведени су у подешавањима. Ова подешавања могу бити ручно преправљена уколико за то постоји потреба, али у већини случајева она су и више него компатибилна за развој потребног пројекта. Аутоконфигурација је једна од најважнијих функционалности Spring Boot-а која је највише допринела популаризовању овог фрејмворка.

Spring Boot користи зависни приступ за додавање и конфигурисање почетних зависности које су потребне за креирање нашег пројекта. На основу онога што ми дефинишемо, Spring Boot одлучује које пакете ће да дода у пројекат и са којим почетним вредностима чиме нам штеди време пошто не морамо да их дефинишемо и инсталирамо мануелно. Овај процес се обично назива иницијализација и начин на који се она обавља описан је у поглављу које говори о креирању Spring Boot пројекта. На пример додавањем Spring Web зависности у наш пројекат, добијамо могућност да креирамо веб апликације са минималним залагањем пошто ће сви пакети (нпр. Apache Tomcat веб сервер) бити аутоматски инсталирани и спремни за коришћење унутар нашег пројекта. Spring Security је још једна јако популарна зависност која аутоматски додаје могућност аутентификације и контроле приступа нашој апликацији. Аутентификација ограничава кориснике који могу да приступе апликацији тако што од сваког корисника захтева валидну комбинацију корисничког имена и лозинке и спречава оне који их не поседују. Корисници који прођу корак аутентификације касније подлежу контролама приступа и неки делови апликације (они који су предвиђени за администраторе) им неће бити доступни.

Самосталност Spring Boot апликација је јако битан фактор зато што оне не зависе од спољашњих веб сервера, већ имају веб сервер који је уграђен у њих у кораку иницијализације (нпр. Tomcat или Netty). Као резултат тога ове апликације се могу покренути једноставно, кликом на “Run” дугме унутар развојног окружења. Уколико не желимо да користимо уграђене веб сервере, доступна је и могућност за конфигурацију и коришћење спољашњег веб сервера.

Као што је наведено, коришћење Spring Boot-а нуди многе погодности и реализација овог пројекта је обављена у доста краћем року него што би то био случај да је коришћен класичан Spring фрејмворк, или неки други Java фрејмворк.

### Maven

Apache Maven је алат за аутоматизацију изградње софтвера и користи се најчешће за Java пројекте али подржава и друге програмске језике као што су C#, Ruby, Scala… Maven помаже при изградњи пројекта и навођењу његових зависности. Изградња самог пројекта заснива се на конвенцијама где једино одступања од подразуменване изградње треба да буду наведена а XML фајл који најчешће носи назив pom, скраћено од Project Object Model, дефинише све зависности, описује софтвер који се изграђује, компоненте, спољашње модуле, редослед изградње, директоријуме и потребне додатке (енгл. plug-in). Пример изгледа pom фајла приказан је на слици 3. Maven динамички преузима потребне Java библиотеке које су дефинисане у pom фајлу са једног од неколико расположивих Maven репозиторијума. Овај кеш преузетих библиотека може бити мануелно ажуриран коришћењем неких библиотека које су креиране у локалним пројектима.



*Слика 3. Изглед pom фајла*

Неке од предности коришћења овог алата су:

* Поједностављује процес изградње пројекта
* Омогућава приступ свим потребним инфорамцијама на једноставан начин
* Процес преузимања Jar фајлова обавља се аутоматски
* Побољшава перформансе пројекта
* Једноставно је додати нове зависности преправком pom фајла
* Велики репозиторијум доступних библиотека

### JavaScript

Xampp је веб сервер пакет отвореног кода који се састоји од Apache HTTP сервера, MariaDB базе података и интерпретера скрипти написаних у PHP и Perl програмским језицима. Обично се користи за тестирање пројеката а како већина постојећих веб сервера користи сличне компоненте као и Xampp прелазак из фазе тестирања на имплементацију у реалним системима је доста олакшан. Само име је акроним за оно што програм омогућава:

X - Означава могућност коришћења на различитим платформама (енгл. Cross-platform)

A - Apache HTTP сервер

M - MariaDB база података

P - Подршка за PHP програмски језик

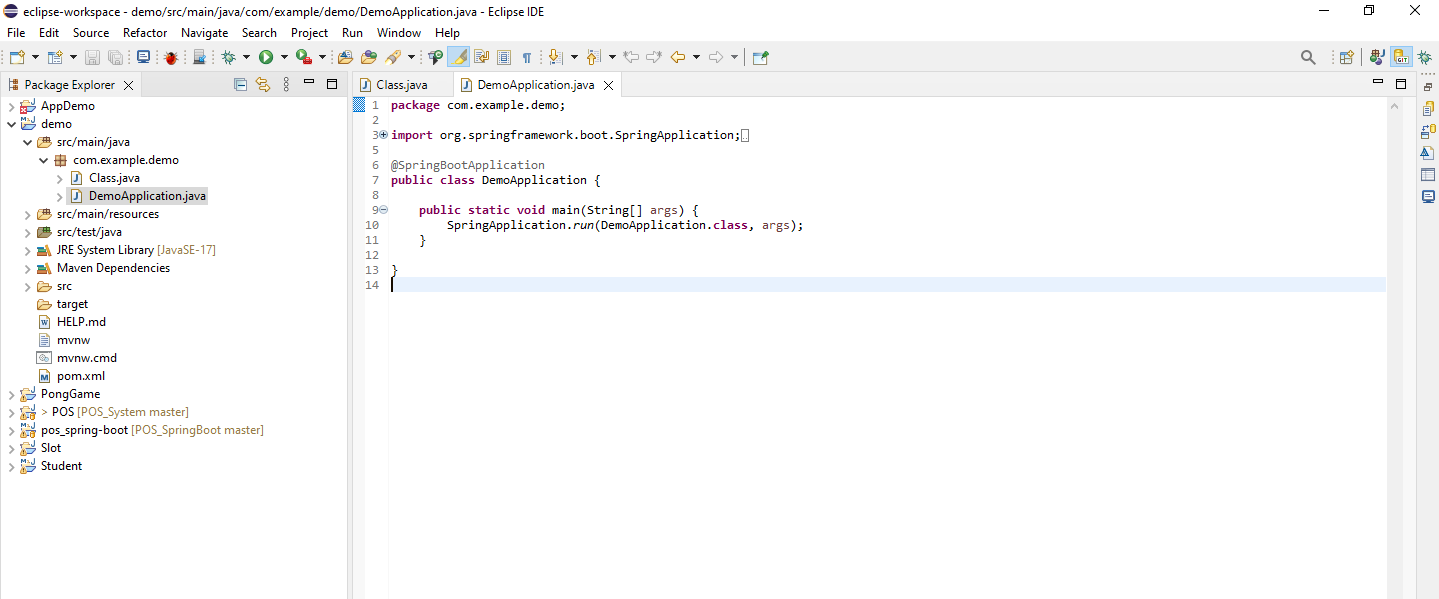
P - Подршка за Perl програмски језик

### Apache Tomcat

Apache Tomcat је још један веб сервер отвореног кода који се користи за покретање “чистог Java кода”. Он функционише тако што пружа HTTP веб сервер окружење у ком је омгућено извршавање и тестирање Java кода. Сервер одржава и развија Apache foundation и он имплементира Java Servlet-е, Java Server Pages, Java Expression Languages и Java Web Socket-е који су минимални захтеви за извршавање Java веб апликације. Како би се апликације могле покренути на Tomcat-у непоходно га је конфигурисати, а коришћењем Spring Boot-а ово конфигурисање обавља се аутоматски и додатна мануелна подешавања нису неопходна.

### InteliJ

Eclipse је интегрисано развојно окружење (енгл. IDE – Integrated Development Environment) коришћено у програмирању. Eclipse је написан у Java програмском језику и користи се за писање програма управо у Javi. Коришћењем додатака може се омогућити и писање програма у другим програмским језицима као што су: C#, C++, Ruby, PHP, Perl, Python, JavaScript и многи други... Састоји се од радне повришине и има систем који омогућава инсталирање додатака којима корисник може да модификује радну површину према жељеном изгледу или дода подршку за неке функционалности које су му потребне за развој пројекта. Java пограмери користе Eclipse уз подршку Eclipse software development kit-а (SDK) који укључује алат потребан за развијање софтвера у Java програмском језику. Последња верзија Eclipse-а је 4.24 и подржава Java верзију 11 или новије. На слици 4. приказан је изглед Eclipse радног окружења.



*Слика 4. Изглед Eclipse радног окружења*

### Thymeleaf

Thymeleaf је програм за динамичко креирање шаблона у Java програмском језику. Може да ради у веб окружењима а и ван њих и пружа потпуну интеграцију са Spring фрејмворком. Користи се као замена за JSP (Јava Server Pages) и имплементира концепт “природних шаблона” што су шаблони које директно можемо отворити у веб прегледачу при чему ће бити приказани као класичне веб странице. Као и већина коришћених програма и Thymeleaf је програм отвореног кода и првенствено је развијен од стране Данијела Фернандеза.

## Коришћене зависности

Као што је већ наведено, при иницијализацији сваког Spring Boot пројекта потребно је навести зависности које ће се користити при развоју. За реализацију овог пројекта коришћене су следеће зависности:

* spring-boot-starter – основна зависност коју мора садржати сваки пројекат
* spring-web – пошто планирамо да креирамо веб апликацију ова зависност позива потребне пакете и омогућава комуникацију у Servlet окружењу
* spring-security – безбедносна зависност која са собом доноси могућност аутентификације корисника и ограничења приступа
* spring-boot-thymeleaf – зависност која омогућава коришћење thymeleaf-а
* spring-boot-data-jpa – омогућава имплементацију репозиторијума што олакшава приступ подацима и њихову манипулацију
* mysql-connector-java – повезивање са базом података
* jasperreports – креирање извештаја и фактура

## Глобални опис

### Перспектива система

Прва верзија система има за циљ да пружи корисницима могућност да генеришу фактуре на основу продатих производа. Такође, биће им омогућено уношење, прегледање и брисање података о купцима и производима као и сторнирање рачуна. Администратори ће имати додатне могућности као што су управљање налозима радника, прегледање рачуна и објављивање порука које се приказују на почетној страни система.

### Карактеристике система

Систем ће све податке преузимати из базе података према томе пре коришћења система потребно је унети све потребне информације у њу. Корисник система ће уносити шифру производа заједно са количином тог производа и на тај начин додавати их на рачун. Свака ставка има своју пореску стопу коју је потребно обрачунати пре наплате и додати на целокупну цену. Након додавања ставки може да се изврши наплата при чему се генерише фактура електронским путем. По генерисању фактуре она се по потреби може одштампати. Начин плаћања је вирмански и подаци о рачуну на који је потребно извршити уплату налазе се на самој фактури. Уколико дође до грешке или се купац предомисли, могуће је сторнирање фактуре кликом на одговарајуће дугме унутар секције за рачуне при чему се врши повраћај уплаћеног новца. Пошто се унутар система чувају осетљиви подаци потребно је обезбедити читав систем тако да се забрани приступ злонамерним корисницима. Дизајн система треба учинти интуитивним тако да и они којима рад за рачунаром није јача страна могу да се снађу и користе систем на начин на који је предвиђен.

### Класе корисника и карактеристике

Апликација разликује две класе корисника: корисник и администратор. Обе класе имају могућност издавања рачуна, генерисања фактура и манипулације подацима о купцима и производима. Додатно, администратори могу и да управљају налозима и објављују поруке на почетној страни система.

## Дефинисање захтева апликације

Захтеви апликације се односе на функционалности које је потребно имплементирати унутар система. Деле се на функционалне, нефункционалне и процесне захтеве. Приоритет захтева означен је бројем, тако да нижи број означава већи приоритет.

### Функционални захтеви

Помоћу функционалних захтева дефинишемо шта систем треба да ради тако што идентификујемо задатке, акције или активности система.

У табели 1. приказани су функционални захтеви за корисника.

*Табела 1. Функционални захтеви за корисника*

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број | Захтев | Опис | Приоритет |
| 1. | Логовање | Систем треба да обезбеди приступ аутентификованом кориснику | 1 |
| 2. | Додавање ставки на рачун | Корисник може да дода жељене ставке на рачун | 1 |
| 3. | Наплаћивање | Корисник може да изврши наплату | 1 |
| 4. | Генерисање фактуре | Корисник моће да генерише фактуру за извршену продају | 1 |
| 5. | Додавање купаца | Додавање купаца у базу података | 1 |
| 6. | Измена података о купцу | Корисник може да мења податке о купцу | 1 |
| 7. | Преглед и претрага купаца | Корисник може да прегледа и претражује купце који се налазе у бази података | 1 |
| 8. | Преглед и претрага производа | Корисник може да прегледа и претражује производе | 1 |
| 9. | Сторнирање | Корисник може да изврши сторнирање рачуна | 1 |
| 10. | Преглед рачуна | Корисник може да прегледа ставке на рачунима | 2 |
| 11. | Преглед новости | Корисник може да прегледа новости | 2 |

У табели 2. приказани су функционални захтеви за администратора.

*Табела 2. Функционални захтеви за администратора*

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број | Захтев | Опис | Приоритет |
| 1. | Логовање | Систем треба да обезбеди приступ администратору | 1 |
| 2. | Уређивање производа | Администратор може да прегледа, додаје, мења и брише производе | 1 |
| 3. | Уређивање купаца | Администратор може да прегледа, додаје, мења и брише купце | 1 |
| 4. | Уређивање корисника | Администратор може да прегледа, додаје, мења и брише кориснике | 1 |
| 5. | Преглед рачуна | Администратор има увид у рачуне | 1 |
| 6. | Издавање и сторнирање рачуна | Администратор може да издаје и сторнира рачуне | 1 |
| 7. | Додавање обавештења | Администратор може да додаје обавештења која ће бити приказана на почетној страни | 2 |

### Нефункционални захтеви

На основу нефункционалних захтева можемо да дефинишемо како би систем требало да функционише. Такође, ови захтеви могу да нам помогну да проценимо успех имплементације дефинисаних функционалних захтева.

У табели 3. приказани су нефункционални захтеви.

*Табела 3. Нефункционални захтеви система*

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број | Захтев | Опис | Приоритет |
| 1. | Перформансе | Систем треба да одговара на корисникове захтеве у што краћем року | 1 |
| 2. | Безбедност | Систем треба да забрани приступ неаутентификованим корисницима | 1 |
| 3. | Ограничење приступа | Корисници не смеју да имају приступ појединим деловима система | 1 |
| 4. | Поузданост | Систем треба да буде доступан 24/7 | 1 |
| 5. | Ажурност | Уколико се на систему открију грешке потребно их је уклонити у што краћем року | 1 |
| 6. | Интуитивност | Навигација кроз систем треба да буде што једноставнија | 1 |
| 7. | Windows подршка | Систем ће подржавати Windows оперативне системе | 1 |
| 8. | Скалабилност | Систем ће подржавати приступ више корисника истовремено | 1 |
| 9. | Дизајн | Систем ће имати савремени дизајн | 2 |
| 10. | Подршка за различите прегледаче | Систему ће моћи да се приступи са различитх веб прегледача | 2 |
| 11. | Техничка подршка корисницима | Корисници ће на располагању имати техничку подршку на даљину | 3 |

### Процесни захтеви

Процесни захтеви се односе на начин на накоји ће систем бити развијен тјст. који програмски језик ће бити коришћен, која архитектура, које технологије ће бити коришћене...

У табели 4. Приказани су процесни захтеви.

*Табела 4. Процесни захтеви система*

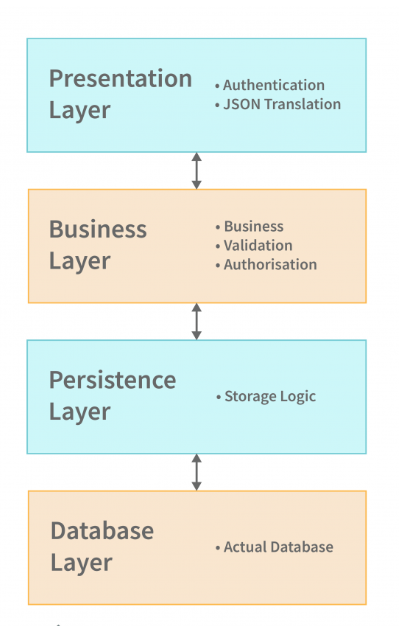
|  |  |  |  |
| --- | --- | --- | --- |
| Редни број | Захтев | Опис | Приоритет |
| 1. | Програмски језик | Апликација ће бити рађена у Java програмском језику | 1 |
| 2. | Фрејмворк | Апликација ће користити Spring Boot фрејмворк | 1 |
| 3. | База података | Апликација ће користити MariaDB базу података, и у њој ће бити чувани сви подаци | 1 |
| 4. | Генерисање фактура | За генерисање фактура користиће се Jasper Reports алат | 1 |
| 5. | Фронтенд | Фронтенд део система биће кодиран у HTML-у уз подршку Bootstrap-a и Thymeleaf-a | 1 |
| 6. | Развојно окружење | За развој апликације користиће се Eclipse развојно окружење | 2 |

# Архитектура апликације

Архитектура апликације има јако важну улогу у њеном развоју и битно је да она буде правилно документована тако да развојни тим који ради на апликацији може да подели процес развоја на делове који се могу радити независно, а касније се они обједињују у једну целину. Потребно је назначити елементе архитектуре, а оне елементе који треба да међусобно комуницирају означити везом. Овим се окалшава и комуникација између чланова тима који раде на развоју пројекта, чувају се битне одлуке о дизајну и омогућава се поновно коришћење креираних елемената. Неправилан приказ архитектуре може довести до неуспешног пројекта и губитка утрошеног времена и ресурса. У наставку ће бити описана архитектура која се користи при развоју Spring Boot апликација.

## Spring Boot архитектура

Spring Boot користи слојевиту архитектуру у којој су слојеви хијерархијски повезани и међусобно комуницирају са слојем испод или изнад њих (слика 5).



*Слика 5. Spring Boot архитектура*

Као што је приказано на слици 5. четири слоја Spring Boot архитектуре су:

* Презентациони слој
* Бизнис слој
* Истрајни слој
* Слој базе података

Презентациони слој налази се на врху хијерархије и он је обично задужен за манипулацију HTTP захтевима (GET, PUT, POST, DELETE) које шаље клијент. Овај слој има могућност да интерпретира JSON (Java Script Object Notion) објекте и врши функционалности као што су аутентификација. У овом слоју такође налазе се и погледи, тј. фронт-енд део апликације. Презентациони слој је еквивалентан са контролер класом у стандардној MVC (Model View Controller) архитектури. Након пријема захтева прелазимо на бизнис слој на даље процесирање.

Бизнис слој је задужен за функционалности као што су валидација, ауторизација и бизнис логика. У софтверској терминологији бизнис логика подразумева део у коме програмери „кажу“ програму шта да ради у одређеним ситуацијама. Нпр. програм дозвољава само администраторима да прегледају податке о корисницима, уколико корисник покуша да приступи подацима о другим корисницима он ће бити спречен.

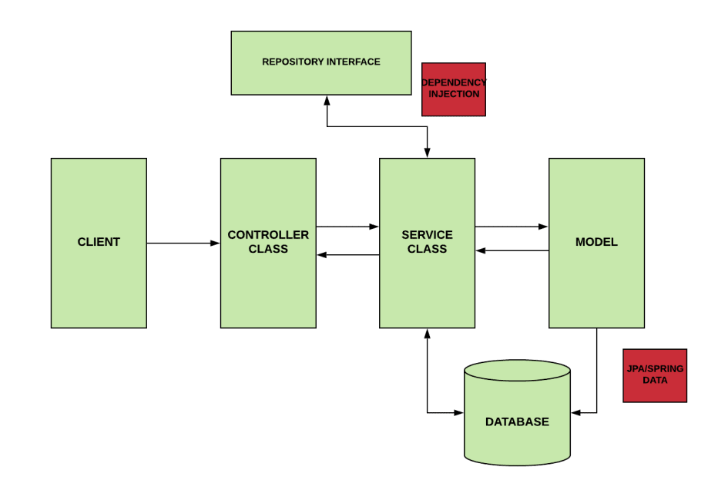
Истрајни слој је слој који комуницира између бизнис слоја и слоја базе података и према томе је задужен за превођење рачунарских објеката у колоне у бази података и обрнуто. У овом слују се дефинишу упити који ће се вршити над базом податка и еквивалентно је са репозиторијским класама у другим архитектурама. Сва складишна логика чува се у овом слоју.

У слоју базе података као што и име наговештава налази се база података, то може бити MySql, MangoDB или било која друга база података. Овај слој може садржати више база одједном и задужен је за CRUD (Create Read Update Delete) операције. Састоји се од технологија које омогућавају нашој мобилној или веб апликацији могућност да чува и преузима податке из базе.

Разлог због које се већина програмера одлучује за ову архитектуру при изради мањих пројеката је њена једноставност. Свака компонента је засебна што олакшава њихово тестирање и проналажење грешака унутар кода. Међутим, пошто ова архитектура захтева доста одржавања зато што промена на једном слоју може да има утицај на све повезане слојеве и према томе постоји аргумент да би она требала да се користи искључиво при изради већих пројеката и унутар већих развојних тимова. Такође, скалабилност пројекта може да представља проблем уколико структура фрејмворка не дозвољава раст. Али без обзира на то, када нам је потребно да неку апликацију развијемо што брже и уколико не поседујемо обимно знање из области софтверских архитектура требало би да користимо неку врсту слојевите архитектуре.

## Spring Boot архитектура радног тока

Архитектура радног тока односи се на ток догађаја који се дешавају након што клијент направи један од HTTP захтева на апликацији. На слици 6. приказана је архитектура радног тока Spring Boot система.

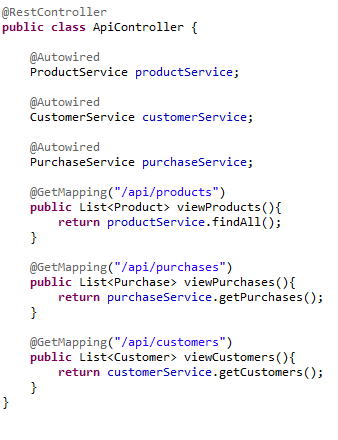


*Слика 6. Spring Boot архитектура радног тока*

Апликација функционише на следећи начин:

1. Клијент креира HTTP захтев
2. Контролер класа прихвата овај захтев
3. Контролер одлучује шта треба урадити са прихваћеним захтевом на основу врсте захтева (POST, PUT, DELETE…)
4. Уколико је потребно позива се сервис класа која даље позива модел класу и комуницира са базом података
5. Сервис класа примењује дефинисану бизнис логику
6. Уколико није дошло да грешке, као резултат захтева враћа се HTML страница.

Контролер класа је једна од најважнијих компоненти и задужена је за обраду захтева, конструкцију модела и рендеровање погледа који се враћа у виду одговора на захтев. Све контролер класе се означавају са @Controller анотацијом или ако је у питању Rest контролер онда је анотација @RestController. Ово омогућава Spring-у да препозна ове класе као контролоре и позове их при доласку захтева. Пример контролера приказан је на слици 7.



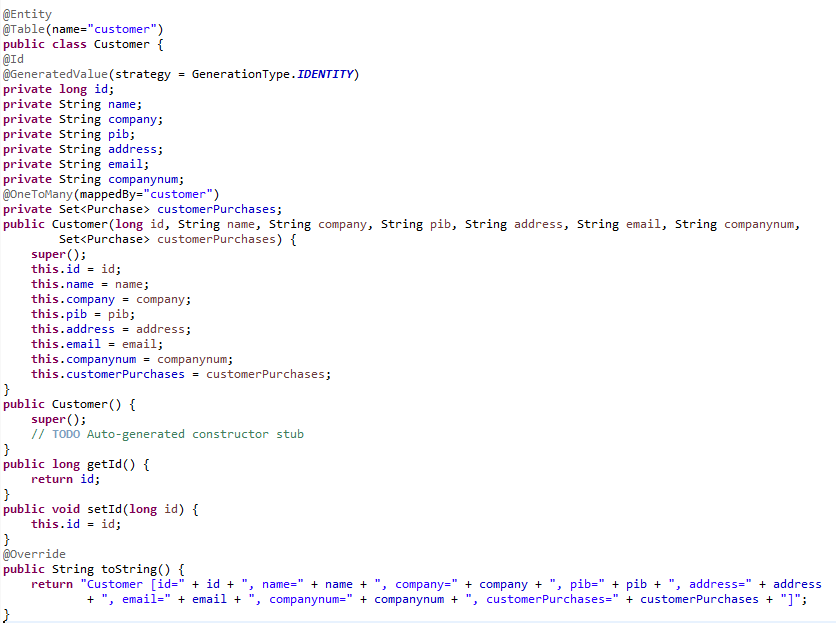
*Слика 7. Контролер класа*

@RestController анотација означава да се ради о Rest контролеру, тј. контролеру који као одговор враћа податке у JSON облику.

@Autowired анотација се користи за аутоматско инјектовање зависности. Инјектовање зависности је техника која се користи у програмирању како би се класа учинила независном. Креирање објеката директно унутар класе није флексибилно зато што читаву класу посвети једном објекту и не дозвољава мењање те инстанце касније у току програма.

@GetMapping анотација означава да се ради о HTTP Get захтеву и унутар заграда потребно је да дефинишемо путању на коју контролер одговара и резултат одговора. У овом случају као одговор добија се листа.

Модел се односи на ентитете који су креирани за потребе апликације. Свака табела унутар базе података представља посебан ентитет (корисник, купац, производ...). Ентитети се у Spring Boot-у означавају са @Entity анотацијом и она је део spring-boot-data-jpa зависности. На слици 8. приказан је пример једне класе која корисити @Entity анотацију.



*Слика 8. Entity класа*

Поред @Entity анотације коришћено је још неколико анотација које су неопходне за програмску логику.

@Table анотација означава назив табеле унутар базе података коју користимо.

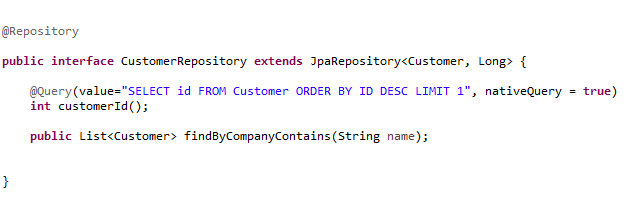
@Id анотација означава да је колона примарни кључ унутар своје табеле и према томе потребно је да поштује одређен скуп правила.

@GeneratedValue анотација дефинише начин на који се генеришу вредности за примарни кључ. У овом случају, то значи да ће вредности за примарни кључ бити аутоматски инкрементоване тј. за сваки нови ред у табели доделиће се примарни кључ чија ће вредност бити за један већа од примарног кључа у претходном реду.

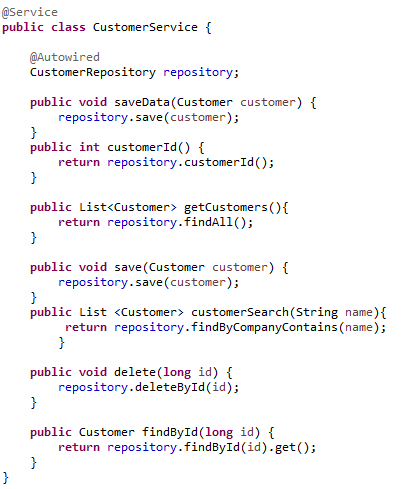
На крају имамо @OneToMany анотацију која означава ће тип логичке везе између две табеле бити један на према више. Поред ове анотације имамо и @OneToOne и @ManyToMany.

Поред наведених анотација саставни део кода је и наравно сама класа заједно са дефинисаним атрибутима, конструкторима и методама унутар ње.

Сервис класе обично функционишу заједно са репозиторијумима и оне у већини случајева имплементирају пословну логику. Ове две компоненте директно комуницирају са базом података. Репозиторијуми су интерфејси који користе @Repository анотацију и унутар њих по потреби можемо дефинисати упите коришћењем @Query анотације или користити неке од предефинисаних метода. Касније унутар сервис класе користимо потребне методе из репозиторијума и по потреби можемо имплементирати пословну логику.



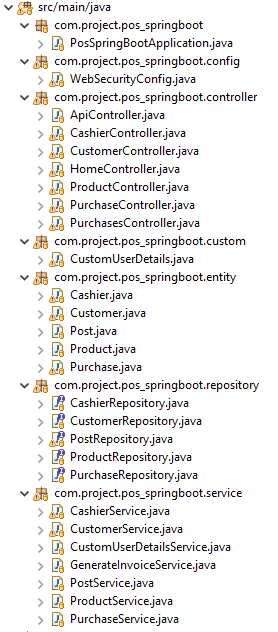
*Слика 9. Пример репозитуријума*



*Слика 10. Пример сервис класе*

Све методе са слике 10. осим customerSearch методе су већ предефинисане унутар репозиторијума. customerSearch метода је метода коју смо ми дефинисали и проследили упит у базу коришћењем @Query анотације. Сврха ове методе је да претражи купце на основу имена компаније.

Унутар нашег пројекта ове компоненте архитектуре су разврстане по пакетима и на слици 11. приказан је њихов изглед.



*Слика 11. Изглед архитектуре унутар пројекта*

Поред описаних компоненти имамо још два пакета са екстензијом Custom и Config унутар ког су дефинисане класе које садрже потребне методе за ауторизацију корисника при логовању на апликацију.

## База података

Као што је наведено у ранијем поглављу користиће се Xampp заједно са MariaDB базом података. Како би могли да се успоставимо конекцију са базом података потребно је дефинисати одређене параметре конекције унутар application.properties фајла. Овај фајл садржи следеће податке:

spring.datasource.url=jdbc:mysql://localhost:8111/pos?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC

server.error.whitelabel.enabled=false

server.port=7050

spring.datasource.username=root

spring.datasource.password=

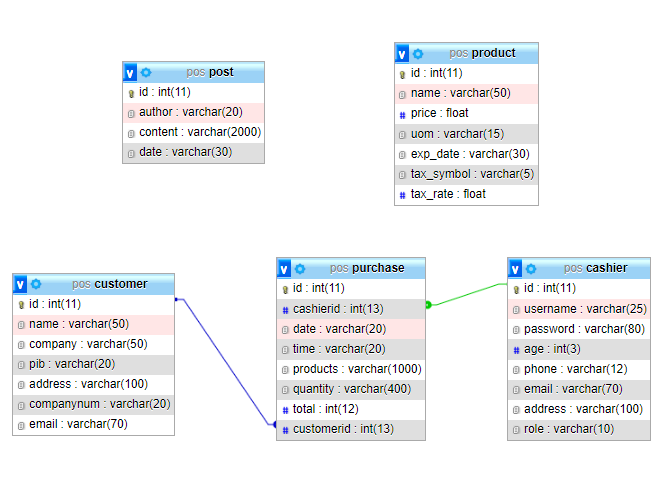
spring.jpa.open-in-view=false

spring.thymeleaf.cache=false

api.base.path = http://localhost:7050

Као што можемо да видимо за базу података користи се MariaDB база лоцирана на хосту и слуша на порту 8111. Порт који се користи за приступ апликацији постављен је да буде 7050. Корисничко име за приступ бази података је root, а шифра није дефинисана што би се генерално сматрало као безбедносни пропуст али у периоду тестирања шифру можемо занемарити. Поред тога искључено је Thymeleaf кеширање података што значи да при сваком покретању странице „свежи” подаци се повлаче из базе. Ова опција може да има лош утицај на перформансе али је јако корисна у фази тестирања пошто не морамо да изнова покрећемо апликацију при свакој промени. Такође, још неке опције су искључене како би се побољшале перформансе апликације.

Сама база података састоји се од 5 табела и 2 релационе везе између ових табела које су приказане на слици 12.

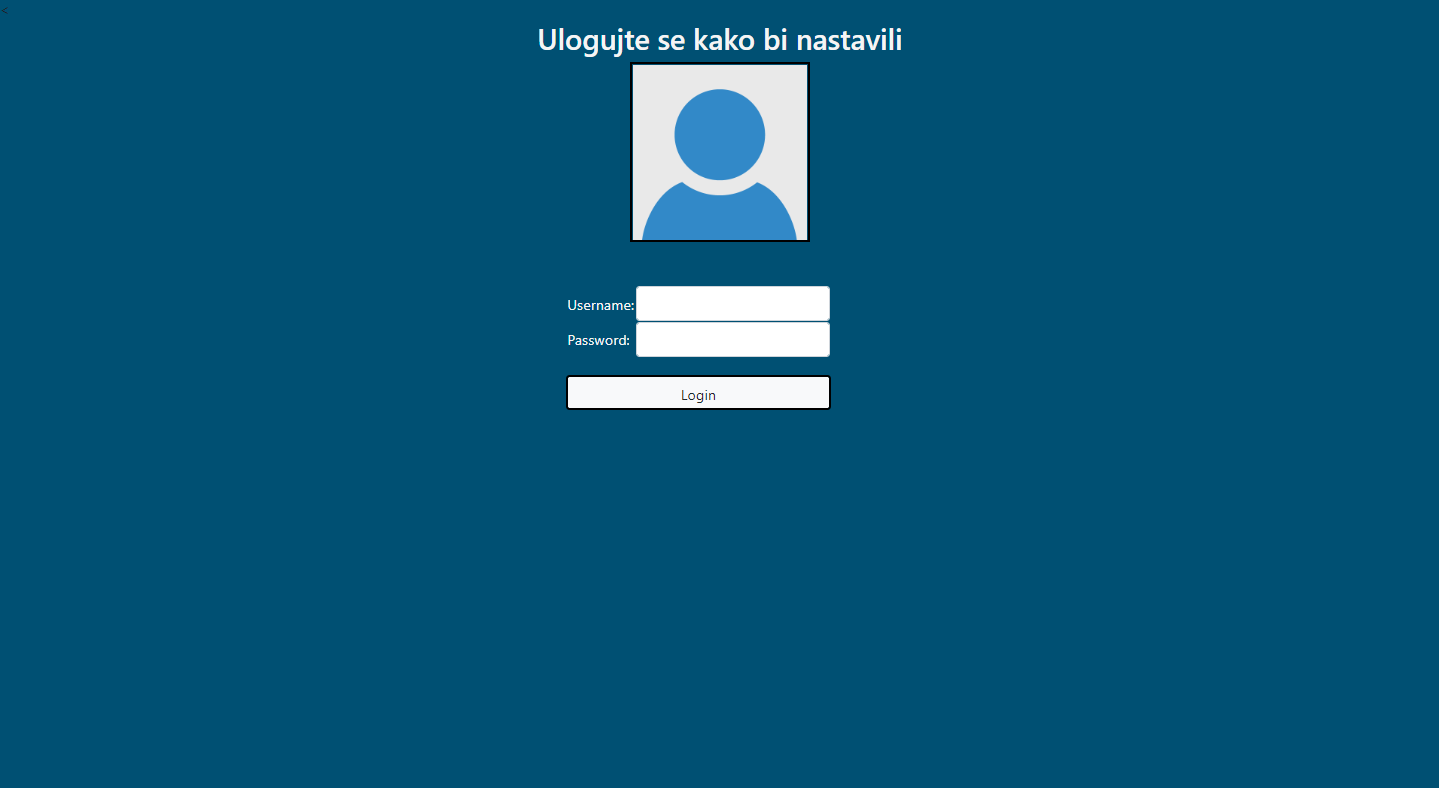


*Слика 12. Структура базе података*

# Изглед апликације

Изглед апликације има јако битну улогу зато што је потребно да сваки корисник, без обзира на искуство које има са радом на рачунару, са лакоћом користи нашу апликацију. Уколико апликација није довољно интуитивна то може довести до забуне корисника и погрешног руковања апликацијом што даље може проузроковати грешке, губитак података и у најгорем случају пад целокупног система. Навигација кроз апликацију мора да буде јасно дефинисана, свако поље за унос података треба да има ознаку помоћу које корисник треба да разуме коју врсту податка треба да унесе, функционалност дугмета треба да буде исписана на самом дугмету и уколико је функционалност тог дугмета неопозива (као нпр. брисање података) онда то дугме треба обојити одређеном бојом како би корисник био упозорен.

При отварању странице, корисник се сусреће са формом за пријаву на систем где треба да унесе своје корисничко име и шифру. На слици 13. приказана је ова форма.



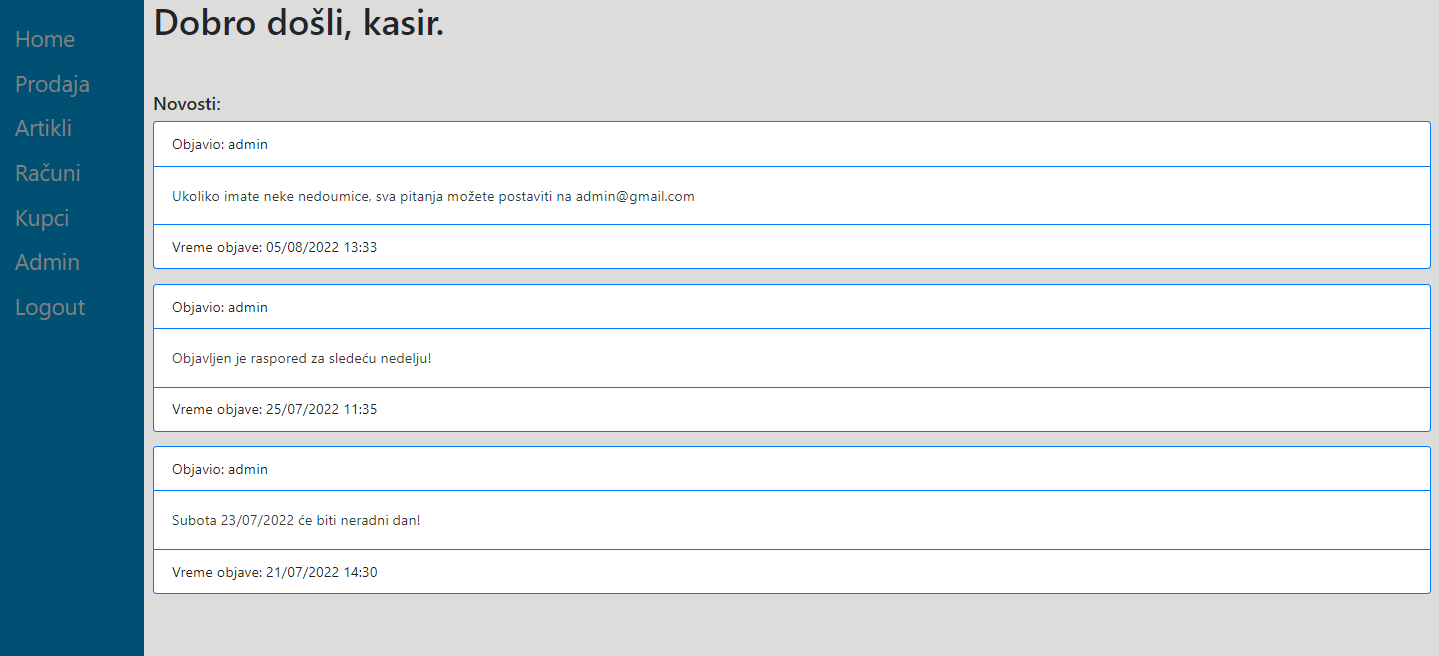
*Слика 13. Страница са формом за логовање*

Систем користи BCrypt енкрипцију како би систем био заштићен од различитих напада. Овај вид енкрипције заснива се на хеш функцији која унесени текст претвара у насумичан низ карактера који су увек исте дужине и такав низ карактера чува се унутар базе података уместо самог текста. Касније при логовању врши се енкрипција унете шифре и уколико се тај хеш поклапа са оним који је унет у бази података кориснику ће бити омогућен приступ систему, у супротном случају логовање неће бити успешно. Уз то, користи се и Salt (насумичан низ карактера који се додаје на текст пре енкрипције) што додаје још један ниво заштите.

На пример за унету шифру ,,password’’ текст који се добија енкрипцијом је: $2a$12$1Bp9mACDpczG5RZIYr8NieQQx6qnX.PyRv3YK/zgKzJrezL/D6QQ6 што би било јако тешко и готово немогуће декриптовати осим ако унапред не знамо како изгледа хеш појединих шифри које се често користе.

У наставку биће приказан изглед сајта из перспективе касира који има улогу обичног корисника.

На слици 14. приказан је изглед почетне стране на коју коју се корисник усмерава након успешног логовања.

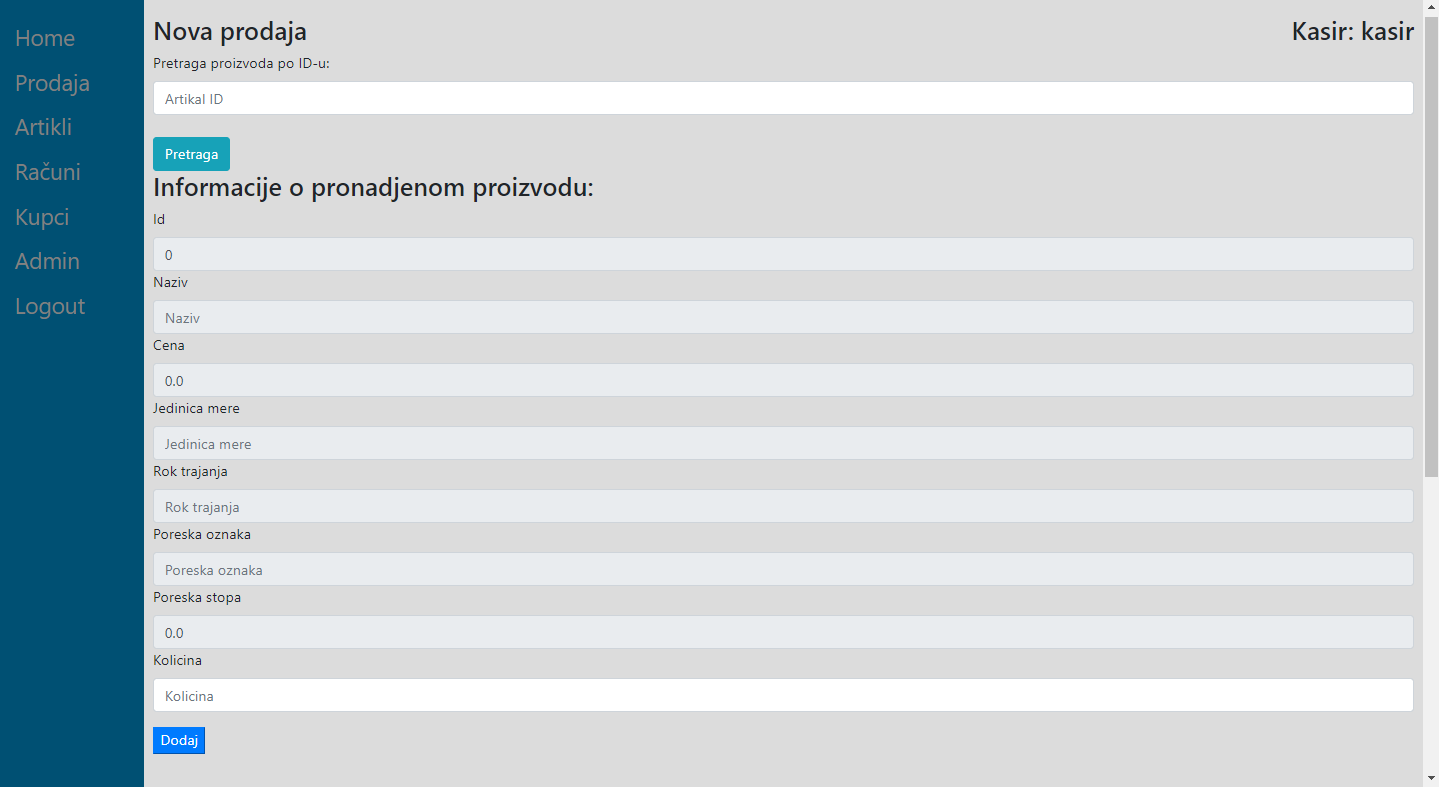


*Слика 14. Изглед почетне стране*

На почетној страни приказана су сва обавештења по хронолошком реду. Навигациона трака налази се са леве стране и унутар ње се налазе следеће ставке:

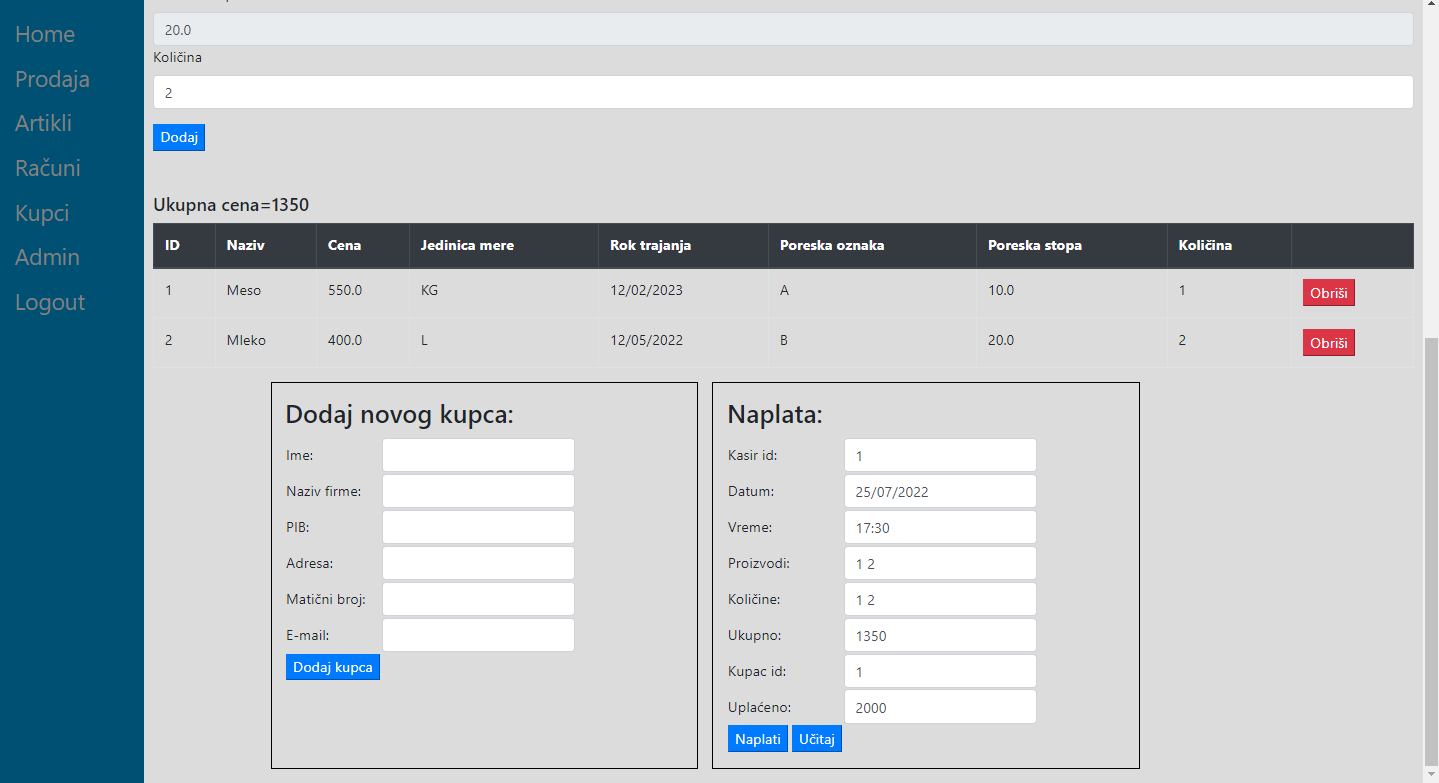
* Home – почетна страна
* Prodaja – креирање нове продаје
* Artikli – преглед артикла
* Računi – преглед рачуна, сторнирање рачуна
* Kupci – преглед купаца
* Admin – посебна страница за администраторе система
* Logout – одјава са апликације

Кликом на ,,Prodaja’’ отвара се страница приказана на слици 15.



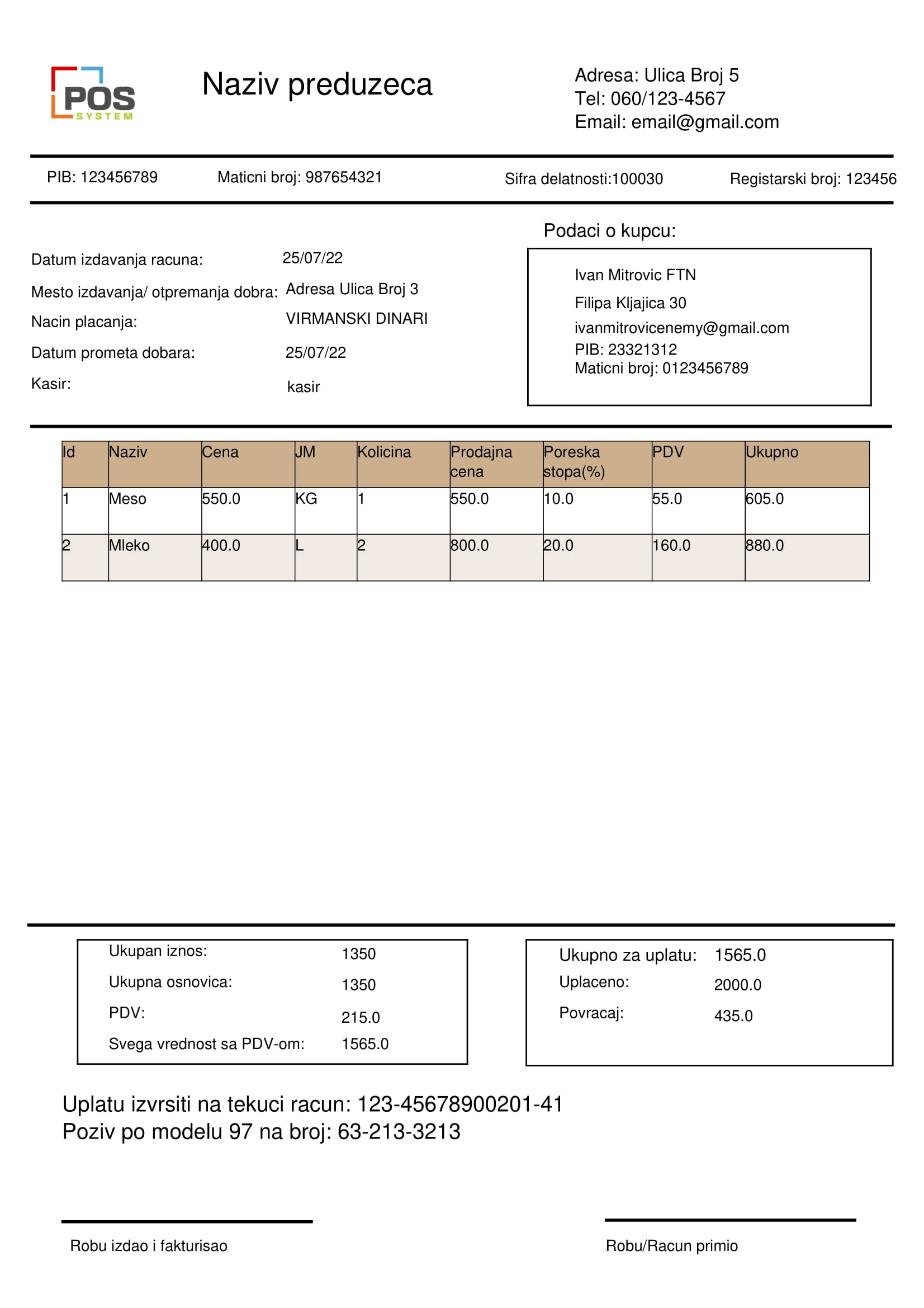
*Слика 15. Страница за продају*

Корисник претражује артикле по њиховом ID-у, уноси њихову количину и на тај начин их додаје на рачун. Након додавања свих потребних арткала може се прећи на наплату која је приказана на слици 16.



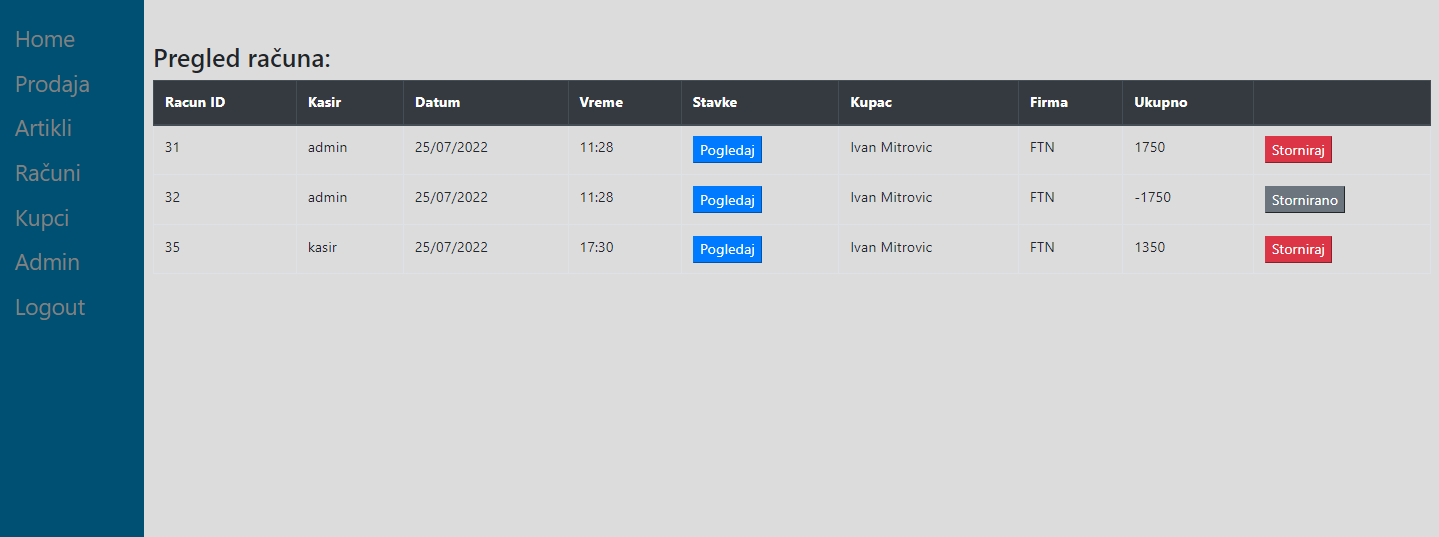
*Слика 16. Страница за продају - део за наплату*

Уколико купац не постоји у бази података можемо га додати. Кликом на дугме ,,Učitaj’’ попуњавају се сви потребни подаци осим поља „Uplaćeno“ где је потребно унети уплаћену количину новца. Кликом на дугме „Naplati“ куповина се чува у бази података и генерише се фактура приказана на слици 17.



*Слика 17. Изглд генерисане фактуре*

Кликом на „Računi“ у навигационом менију отвара се страница приказана на слици 18.



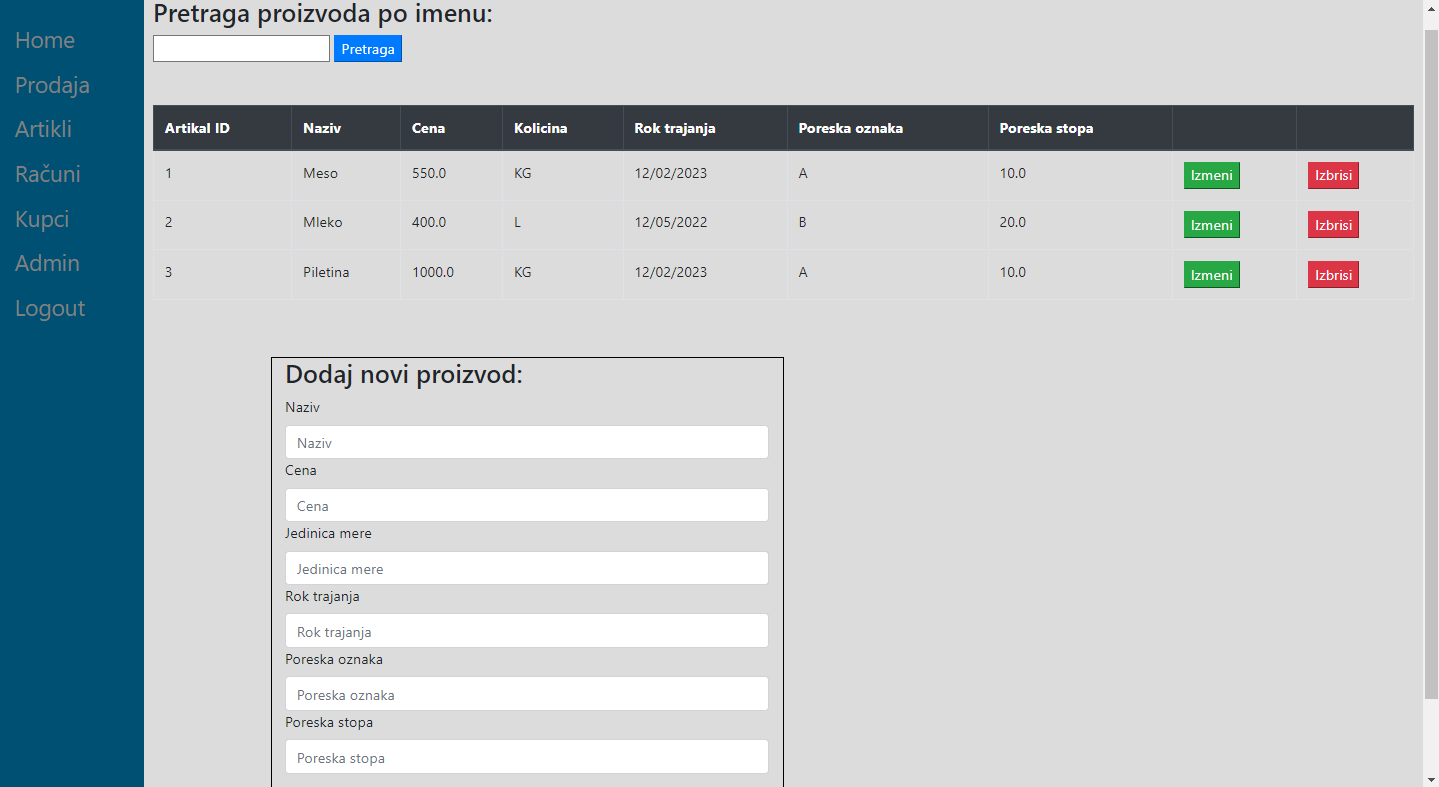
*Слика 18. Страница за преглед рачуна*

На овој страници имамо увид у све издате рачуне и уколико желимо, кликом на дугме „Storniraj“ можемо да извршимо сторнирање рачуна уколико он није већ сторниран. Кликом на дугме „Pogledaj“ унутар колоне „Stavke“ бићемо пребачени на страницу на којој можемо да погледамо ставке са рачуна. Ова страница приказана је на слици 18.



*Слика 19. Преглед ставки са рачуна*

Кликом на „Artikli“ унутар навигационог менија отвара се страница где можемо да прегледамо све артикле унутар базе система. На слици 19. приказана је страница за преглед артикала.



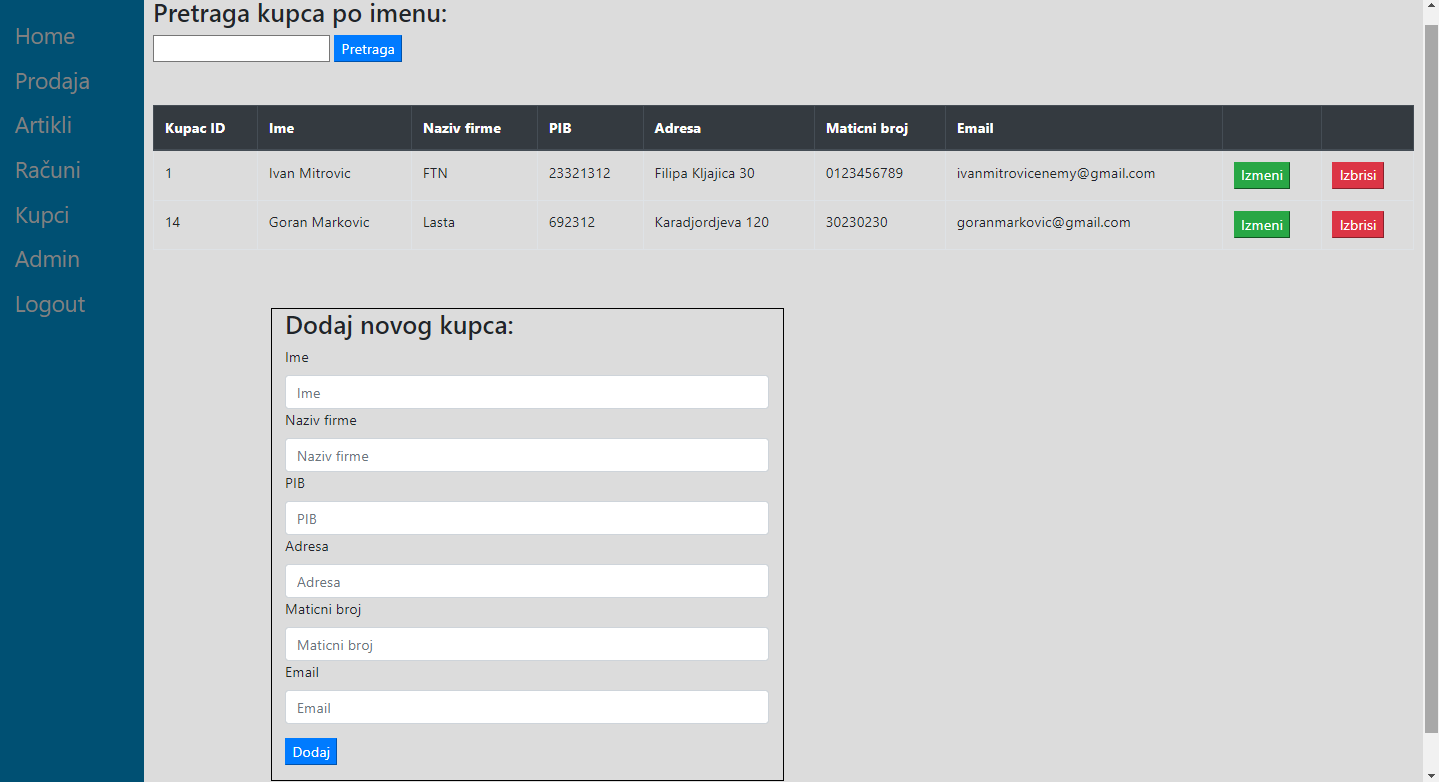
*Слика 20. Страница „Artikli“*

Такође на овој страници имамо и опцију за додавање новог артикла и измену или брисање постојећег артикла. Уколико желимо да обришемо артикал потребно је да кликнемо на дугме „Obriši“, а уколико желимо да изменимо неки од артикала то можемо учинити кликом на дугме „Izmeni“ при чему се отвара нова страница приказана на слици 21.



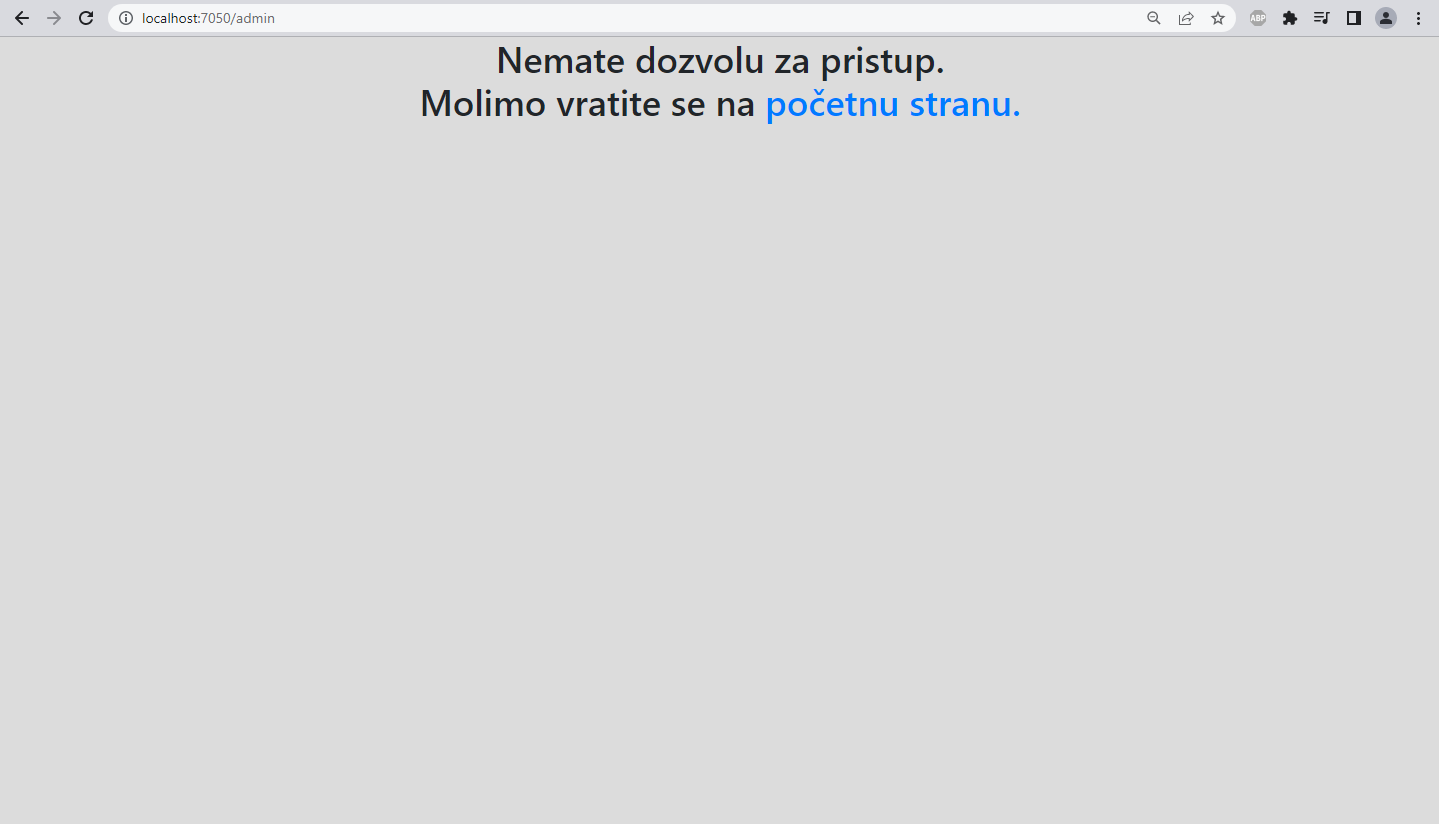
*Слика 21. Страница за измену артикла*

Страница „Kupci“ функционише на сличан начин као и страница са артиклима и она је приказана на слици 21.



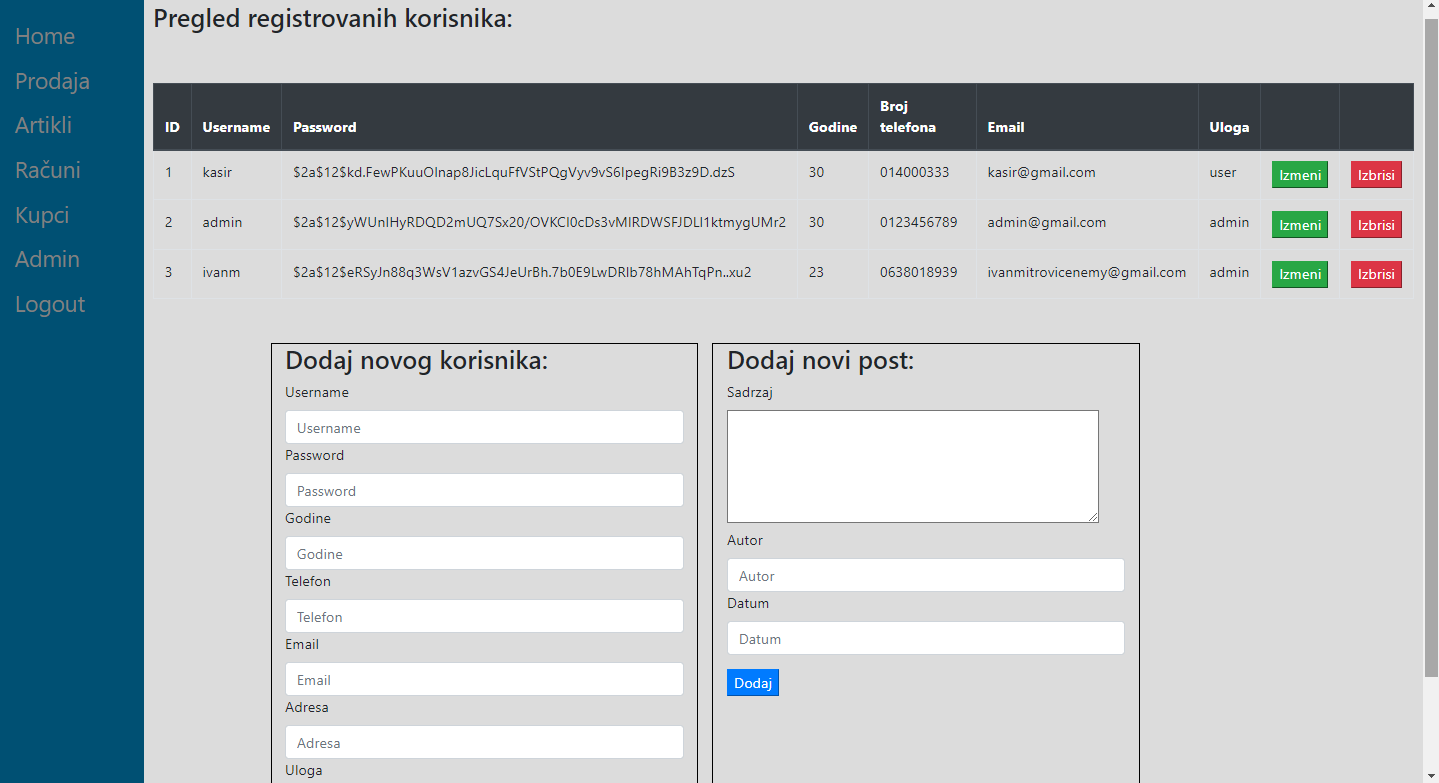
*Слика 22. Страница „Kupci“*

Како смо тренутно улоговани као обичан корисник уколико покушамо да отворимо страницу „Admin“ добићемо грешку приказану на слици 22.



*Слика 23. Грешка при отварању „Admin“ странице*

Након преласка на налог који има улогу админа приступ овој страници биће нам омогућен и њен изглед приказан је на слици 24.



*Слика 24. „Admin“ страница*

На овој страници имамо преглед регистрованих корисника, могућност измене њихових података и брисања њихових налога. Такође можемо и да додамо новог корисника или ново обавештење које ће бити приказано на почетној страници.

# Закључак

Стално се воде дебате између програмера који програмски језик је најбољи и за коју примену. Без обзира који програмски језик одаберемо да научимо, ако желимо да тим језиком располажемо на високом нивоу, потребно је да научимо и неки од његових фрејмворка. Из године у годину они постају све моћнији и једноставнији за употребу а према томе расте и њихова популарност.

Што се тиче Java програмског језика Spring фрејмворк и конкретно Spring Boot верзија овог фрејмворка је далеко најпопуларнија и учећи како она функционише већина програмера ће схватити и зашто. Користећи овај фрејмворк штеди се доста на времену проведеном кодирајући зато што на располагању имамо велики број раличитих библиотека које можемо да имплементирамо у код апликације, и ове бибилиотеке покривају готово сваки могући захтев који пројекат може да има. Аутентификација, ауторизација, комуникација са базом података, генерисање фактура све ове функционалности и много друге су нам на располагању када се пројекат развија у овом фрејмворку.

На први поглед архитектура Spring Boot фрејмворка може да буде помало конфузна уколико нисмо имали ранијег искуства са сличним фрејмворцима или Java програмским језиком. Потребно је добро савладати зависности, анотације и објектно-оријентисано програмирање да бисмо без проблема могли да развијамо веб апликације коришћењем Spring Boot фрејмворка.

Како би демонстрирали архитектуру и функционалност Spring Boot фрејмворка креирана је једноставна POS апликација. Неки од основих задатака апликација овог типа су преглед и модификација артикала, додавање нових артикала, издавање рачуна, преглед и модификација купаца, додавање нових купаца... Све ове, и неколико других функционалности су успешно имплементиране и тестиране. Апликација има доста простора за надоградњу и неке од будућих верзија могле би да додају могућност креирања извештаја или да додатно побољшају безбедност апликације. Иако је на безбедност утрошено доста времена при развоју, она је од кључног значаја зато што се у апликацији чува доста личних података о корисницима и купцима као и код осталих апликација овог типа.

# Литература

1. Spring Boot Reference Documentantion, преузето са: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>, приступано јула 2022.
2. Greg L. Turnquist: Learning Spring Boot 2.0 Second Edition: Simplify the development of lightning fast applications based on microservices and reactive programming, Novembar 2017, 370p.
3. Dinesh Rajput: Mastering Spring Boot 2.0: Build modern, cloud-native, and distributed systems using Spring Boot, Maj 2018, 380p.
4. Joshua Block: Effective Java: Second Edition, 2004, 262p.
5. Learn Spring Boot, преузето са: <https://www.baeldung.com/spring-boot>, приступано јула 202.
6. Spring Boot – Quick Guide, преузето са: <https://www.tutorialspoint.com/spring_boot/spring_boot_quick_guide.htm>, приступано јула 2022.
7. JasperReports Documentation, преузето са: <https://community.jaspersoft.com/documentation?version=61916>, приступано јула 2022.
8. Top 10 Java Frameworks to Use in 2022, преузето са: <https://hackr.io/blog/java-frameworks>, приступано јула 2022.
9. Securing a Web Application, преузето са: <https://spring.io/guides/gs/securing-web/>, приступано јула 2022.
10. Learn Java, преузето са: <https://dev.java/learn/>, приступано јула 2022.

# Списак слика

[*Слика 1. Spring Initializr* 8](#_Toc111817976)

[*Слика 2. Изглед пројекта након импорта* 8](#_Toc111817977)

[*Слика 3. Изглед pom фајла* 10](#_Toc111817978)

[*Слика 4. Изглед Eclipse радног окружења* 12](#_Toc111817979)

[*Слика 5. Spring Boot архитектура* 18](#_Toc111817980)

[*Слика 6. Spring Boot архитектура радног тока* 20](#_Toc111817981)

[*Слика 7. Контролер класа* 21](#_Toc111817982)

[*Слика 8. Entity класа* 22](#_Toc111817983)

[*Слика 9. Пример репозитуријума* 23](#_Toc111817984)

[*Слика 10. Пример сервис класе* 23](#_Toc111817985)

[*Слика 11. Изглед архитектуре унутар пројекта* 24](#_Toc111817986)

[*Слика 12. Структура базе података* 26](#_Toc111817987)

[*Слика 13. Страница са формом за логовање* 27](#_Toc111817988)

[*Слика 14. Изглед почетне стране* 28](#_Toc111817989)

[*Слика 15. Страница за продају* 29](#_Toc111817990)

[*Слика 16. Страница за продају - део за наплату* 29](#_Toc111817991)

[*Слика 17. Изглд генерисане фактуре* 30](#_Toc111817992)

[*Слика 18. Страница за преглед рачуна* 31](#_Toc111817993)

[*Слика 19. Преглед ставки са рачуна* 31](#_Toc111817994)

[*Слика 20. Страница „Artikli“* 32](#_Toc111817995)

[*Слика 21. Страница за измену артикла* 32](#_Toc111817996)

[*Слика 22. Страница „Kupci“* 33](#_Toc111817997)

[*Слика 23. Грешка при отварању „Admin“ странице* 33](#_Toc111817998)

[*Слика 24. „Admin“ страница* 34](#_Toc111817999)